





6 Tidy Data

6.1 Learning Objectives

Basic

1. Understand the concept of [tidy data](#) 
2. Be able to convert between long and wide formats using pivot functions 
 - [pivot_longer\(\)](#)
 - [pivot_wider\(\)](#)
3. Be able to use the 4 basic tidyr verbs 
 - [gather\(\)](#)
 - [separate\(\)](#)
 - [spread\(\)](#)
 - [unite\(\)](#)
4. Be able to chain functions using [pipes](#) 



Intermediate

5. Be able to use [regular expressions](#) to separate complex columns

6.2 Setup

```
# libraries needed
library(tidyverse)
library(reprores)

set.seed(8675309) # makes sure random numbers are reproducible
```

6.3 Tidy Data

6.3.1 Three Rules

- Each [variable](#) must have its own column
- Each [observation](#) must have its own row
- Each [value](#) must have its own cell

This table has three observations per row and the `total_meanRT` column contains two values.

id	score_1	score_2	score_3	rt_1	rt_2	rt_3	total_meanRT
1	4	3	7	857	890	859	14 (869)
2	3	1	1	902	900	959	5 (920)
3	2	5	4	757	823	901	11 (827)
4	6	2	6	844	788	624	14 (752)
5	1	7	2	659	764	690	10 (704)

Table 6.1: Untidy table

This is the tidy version.

id	trial	rt	score	total	mean_rt
1	1	857	4	14	869
1	2	890	3	14	869
1	3	859	7	14	869
2	1	902	3	5	920
2	2	900	1	5	920
2	3	959	1	5	920
3	1	757	2	11	827
3	2	823	5	11	827
3	3	901	4	11	827
4	1	844	6	14	752
4	2	788	2	14	752
4	3	624	6	14	752
5	1	659	1	10	704
5	2	764	7	10	704
5	3	690	2	10	704

Table 6.1: Tidy table

6.3.2 Wide versus long

Data tables can be in [wide](#) format or [long](#) format (and sometimes a mix of the two). Wide data are where all of the observations about one subject are in the same row, while long data are where each observation is on a separate row. You often need to convert between these formats to do different types of analyses or data processing.

Imagine a study where each subject completes a questionnaire with three items. Each answer is an [observation](#) of that subject. You are probably most familiar with data like this in a wide format, where the subject `id` is in one column, and each of the three item responses is in its own column.

id	Q1	Q2	Q3
A	1	2	3
B	4	5	6

Table 6.2: Wide data

The same data can be represented in a long format by creating a new column that specifies what `item` the observation is from and a new column that specifies the `value` of that observation.

id	item	value
A	Q1	1
B	Q1	4
A	Q2	2
B	Q2	5
A	Q3	3
B	Q3	6

Table 6.3: Long data

Create a long version of the following table.

id	fav_colour	fav_animal
Lisa	red	echidna
Robbie	orange	babirusa
Steven	green	frog

Answer

6.4 Pivot Functions

The pivot functions allow you to transform a data table from wide to long or long to wide in one step.

6.4.1 Load Data

We will use the dataset `personality` from the `reprores` package (or download the data from [personality.csv](#)). These data are from a 5-factor (personality) personality questionnaire. Each question is labelled with the domain (Op = openness, Co = conscientiousness, Ex = extroversion, Ag = agreeableness, and Ne = neuroticism) and the question number.

```
data("personality", package = "reprores")
```

user_id	date	Op1	Ne1	Ne2	Op2	Ex1	Ex2	Co1	Co2	Ne3	Ag1	Ag2	Ne4	Ex3
0	2006-03-23	3	4	0	6	3	3	3	3	0	2	1	3	3
1	2006-02-08	6	0	6	0	0	0	0	0	0	0	6	6	6
2	2005-10-24	6	0	6	0	0	0	0	0	0	0	6	6	5
5	2005-12-07	6	4	4	4	2	3	3	3	1	4	0	2	5
8	2006-07-27	6	1	2	6	2	3	5	4	0	6	5	3	3
108	2006-02-28	3	2	1	4	4	4	4	3	1	5	4	2	3

6.4.2 pivot_longer()

`pivot_longer()` converts a wide data table to long format by converting the headers from specified columns into the values of new columns, and combining the values of those columns into a new condensed column.

- `cols` refers to the columns you want to make long. You can refer to them by their names, like `col1`, `col2`, `col3`, `col4` or `col1:col4` or by their numbers, like `8`, `9`, `10` or `8:10`.

- `names_to` is what you want to call the new columns that the gathered column headers will go into; it's "domain" and "qnumber" in this example.
- `names_sep` is an optional argument if you have more than one value for `names_to`. It specifies the characters or position to split the values of the `cols` headers.
- `values_to` is what you want to call the values in the columns ... ; they're "score" in this example.

```
personality_long <- pivot_longer(
  data = personality,
  cols = Op1:Ex9,                # columns to make long
  names_to = c("domain", "qnumber"), # new column names for headers
  names_sep = 2,                  # how to split the headers
  values_to = "score"             # new column name for values
) %>%
  glimpse()
```

```
## Rows: 615,000
## Columns: 5
## $ user_id <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0,...
## $ date <date> 2006-03-23, 2006-03-23, 2006-03-23, 2006-03-23,
## 2006-03-23, 2...
## $ domain <chr> "Op", "Ne", "Ne", "Op", "Ex", "Ex", "Co", "Co",
## "Ne", "Ag", "A...
## $ qnumber <chr> "1", "1", "2", "2", "1", "2", "1", "2", "3", "1",
## "2", "4", "3...
## $ score <dbl> 3, 4, 0, 6, 3, 3, 3, 3, 0, 2, 1, 3, 3, 2, 2, 1, 3,
## 3, 1, 3, 0,...
```

You can pipe a data table to `glimpse()` at the end to have a quick look at it. It will still save to the object.

What would you set `names_sep` to in order to split the `cols` headers listed below into the results?

cols	names_to	names_sep
A_1, A_2, B_1, B_2	c("condition", "version")	<input type="text" value="2"/>
A1, A2, B1, B2	c("condition", "version")	<input type="text" value="2"/>

```
cat-day&pre, cat-day&post, cat- c("pet",
night&pre, cat-night&post, dog- "time",
day&pre, dog-day&post, dog- "condition")
night&pre, dog-night&post
```

6.4.3 pivot_wider()

We can also go from long to wide format using the `pivot_wider()` function.

- `names_from` is the columns that contain your new column headers.
- `values_from` is the column that contains the values for the new columns.
- `names_sep` is the character string used to join names if `names_from` is more than one column.

```
personality_wide <- pivot_wider(
  data = personality_long,
  names_from = c(domain, qnumber),
  values_from = score,
  names_sep = ""
) %>%
  glimpse()
```

```
## Rows: 15,000
## Columns: 43
## $ user_id <dbl> 0, 1, 2, 5, 8, 108, 233, 298, 426, 436, 685, 807,
## 871, 881, 94...
## $ date <date> 2006-03-23, 2006-02-08, 2005-10-24, 2005-12-07,
## 2006-07-27, 2...
## $ Op1 <dbl> 3, 6, 6, 6, 6, 3, 3, 6, 6, 3, 4, 5, 5, 5, 6, 4, 1,
## 2, 5, 6, 4,...
## $ Ne1 <dbl> 4, 0, 0, 4, 1, 2, 3, 4, 0, 3, 3, 3, 2, 1, 1, 3, 4,
## 5, 2, 4, 5,...
## $ Ne2 <dbl> 0, 6, 6, 4, 2, 1, 2, 3, 1, 2, 5, 5, 3, 1, 1, 1, 1,
## 6, 1, 2, 5,...
## $ Op2 <dbl> 6, 0, 0, 4, 6, 4, 4, 0, 0, 3, 4, 3, 3, 4, 5, 3, 3,
## 4, 1, 6, 6,...
## $ Ex1 <dbl> 3, 0, 0, 2, 2, 4, 4, 3, 5, 4, 1, 1, 3, 3, 1, 3, 5,
## 1, 0, 4, 1,...
## $ Ex2 <dbl> 3, 0, 0, 3, 3, 4, 5, 2, 5, 3, 4, 1, 3, 2, 1, 6, 5,
## 3, 4, 4, 1,...
## $ Co1 <dbl> 3, 0, 0, 3, 5, 4, 3, 4, 5, 3, 3, 3, 1, 5, 5, 4, 4,
## 5, 6, 4, 2,...
## $ Co2 <dbl> 3, 0, 0, 3, 4, 3, 3, 4, 5, 3, 5, 3, 3, 4, 5, 1, 5,
## 4, 5, 2, 5,...
## $ Ne3 <dbl> 0, 0, 0, 1, 0, 1, 4, 4, 0, 4, 2, 5, 1, 2, 5, 5, 2,
```

```

2, 1, 2, 5,...
## $ Ag1      <dbl> 2, 0, 0, 4, 6, 5, 5, 4, 2, 5, 4, 3, 2, 4, 5, 3, 5,
5, 5, 4, 4,...
## $ Ag2      <dbl> 1, 6, 6, 0, 5, 4, 5, 3, 4, 3, 5, 1, 5, 4, 2, 6, 5,
5, 5, 5, 2,...
## $ Ne4      <dbl> 3, 6, 6, 2, 3, 2, 3, 3, 0, 4, 4, 5, 5, 4, 5, 3, 2,
5, 2, 4, 5,...
## $ Ex3      <dbl> 3, 6, 5, 5, 3, 3, 3, 0, 6, 1, 4, 2, 3, 2, 1, 2, 5,
1, 0, 5, 5,...
## $ Co3      <dbl> 2, 0, 1, 3, 4, 4, 5, 4, 5, 3, 4, 3, 4, 4, 5, 4, 2,
4, 5, 2, 2,...
## $ Op3      <dbl> 2, 6, 5, 5, 5, 4, 3, 2, 4, 3, 3, 6, 5, 5, 6, 5, 4,
4, 3, 6, 5,...
## $ Ex4      <dbl> 1, 0, 1, 3, 3, 3, 4, 3, 5, 3, 2, 0, 3, 3, 1, 2, NA,
4, 4, 4, 1...
## $ Op4      <dbl> 3, 0, 1, 6, 6, 3, 3, 0, 6, 3, 4, 5, 4, 5, 6, 6, 2,
2, 4, 5, 5,...
## $ Ex5      <dbl> 3, 0, 1, 6, 3, 3, 4, 2, 5, 2, 2, 4, 2, 3, 0, 4, 5,
2, 3, 1, 1,...
## $ Ag3      <dbl> 1, 0, 1, 1, 0, 4, 4, 4, 3, 3, 4, 4, 3, 4, 4, 5, 5,
4, 5, 3, 4,...
## $ Co4      <dbl> 3, 6, 5, 5, 5, 3, 2, 4, 3, 1, 4, 3, 1, 2, 4, 2, NA,
5, 6, 1, 1...
## $ Co5      <dbl> 0, 6, 5, 5, 5, 3, 3, 1, 5, 1, 2, 4, 4, 4, 2, 1, 6,
4, 3, 1, 3,...
## $ Ne5      <dbl> 3, 0, 1, 4, 1, 1, 4, 5, 0, 3, 4, 6, 2, 0, 1, 1, 0,
4, 3, 1, 5,...
## $ Op5      <dbl> 6, 6, 5, 2, 5, 4, 3, 2, 6, 6, 2, 4, 3, 4, 6, 6, 6,
5, 3, 3, 5,...
## $ Ag4      <dbl> 1, 0, 1, 4, 6, 5, 5, 6, 6, 6, 4, 2, 4, 5, 4, 5, 6,
4, 5, 6, 5,...
## $ Op6      <dbl> 0, 6, 5, 1, 6, 4, 6, 0, 0, 3, 5, 3, 5, 5, 5, 2, 5,
1, 1, 6, 2,...
## $ Co6      <dbl> 6, 0, 1, 4, 6, 5, 6, 5, 4, 3, 5, 5, 4, 6, 6, 1, 3,
4, 5, 4, 6,...
## $ Ex6      <dbl> 3, 6, 5, 3, 0, 4, 3, 1, 6, 3, 2, 1, 4, 2, 1, 5, 6,
2, 1, 2, 1,...
## $ Ne6      <dbl> 1, 6, 5, 1, 0, 1, 3, 4, 0, 4, 4, 5, 2, 1, 5, 6, 1,
2, 2, 3, 5,...
## $ Co7      <dbl> 3, 6, 5, 1, 3, 4, NA, 2, 3, 3, 2, 2, 4, 2, 5, 2, 5,
5, 3, 1, 1...
## $ Ag5      <dbl> 3, 6, 5, 0, 2, 5, 6, 2, 2, 3, 4, 1, 3, 5, 2, 6, 5,
6, 5, 3, 3,...
## $ Co8      <dbl> 3, 0, 1, 1, 3, 4, 3, 0, 1, 3, 2, 2, 1, 2, 4, 3, 2,
4, 5, 2, 6,...
## $ Ex7      <dbl> 3, 6, 5, 4, 1, 2, 5, 3, 6, 3, 4, 3, 5, 1, 1, 6, 6,
3, 1, 1, 3,...

```

```
## $ Ne7      <dbl> NA, 0, 1, 2, 0, 2, 4, 4, 0, 3, 2, 5, 1, 2, 5, 2, 2,
4, 1, 3, 5...
## $ Co9      <dbl> 3, 6, 5, 4, 3, 4, 5, 3, 5, 3, 4, 3, 4, 4, 2, 4, 6,
5, 5, 2, 2,...
## $ Op7      <dbl> 0, 6, 5, 5, 5, 4, 6, 2, 1, 3, 2, 4, 5, 5, 6, 3, 6,
5, 2, 6, 5,...
## $ Ne8      <dbl> 2, 0, 1, 1, 1, 1, 5, 4, 0, 4, 4, 5, 1, 2, 5, 2, 1,
5, 1, 2, 5,...
## $ Ag6      <dbl> NA, 6, 5, 2, 3, 4, 5, 6, 1, 3, 4, 2, 3, 5, 1, 6, 2,
6, 6, 5, 3...
## $ Ag7      <dbl> 3, 0, 1, 1, 1, 3, 3, 5, 0, 3, 2, 1, 2, 3, 5, 6, 4,
4, 6, 6, 2,...
## $ Co10     <dbl> 1, 6, 5, 5, 3, 5, 1, 2, 5, 2, 4, 3, 4, 4, 3, 2, 5,
5, 5, 2, 2,...
## $ Ex8      <dbl> 2, 0, 1, 4, 3, 4, 2, 4, 6, 2, 4, 0, 4, 4, 1, 3, 5,
4, 3, 1, 1,...
## $ Ex9      <dbl> 4, 6, 5, 5, 5, 2, 3, 3, 6, 3, 3, 4, 4, 3, 2, 5, 5,
4, 4, 0, 4,...
```

6.5 Tidy Verbs

The pivot functions above are relatively new functions that combine the four basic tidy verbs. You can also convert data between long and wide formats using these functions. Many researchers still use these functions and older code will not use the pivot functions, so it is useful to know how to interpret these.

6.5.1 `gather()`

Much like `pivot_longer()`, `gather()` makes a wide data table long by creating a column for the headers and a column for the values. The main difference is that you cannot turn the headers into more than one column.

- `key` is what you want to call the new column that the gathered column headers will go into; it's "question" in this example. It is like `names_to` in `pivot_longer()`, but can only take one value (multiple values need to be separated after `separate()`).
- `value` is what you want to call the values in the gathered columns; they're "score" in this example. It is like `values_to` in `pivot_longer()`.
- `...` refers to the columns you want to gather. It is like `cols` in `pivot_longer()`.

The `gather()` function converts `personality` from a wide data table to long format, with a row for each user/question observation. The resulting data table should have the columns: `user_id`, `date`, `question`, and `score`.


```

personality_gathered <- gather(
  data = personality,
  key = "question", # new column name for gathered headers
  value = "score", # new column name for gathered values
  Op1:Ex9          # columns to gather
) %>%
  glimpse()

```

```

## Rows: 615,000
## Columns: 4
## $ user_id <dbl> 0, 1, 2, 5, 8, 108, 233, 298, 426, 436, 685, 807,
871, 881, 9...
## $ date <date> 2006-03-23, 2006-02-08, 2005-10-24, 2005-12-07,
2006-07-27, ...
## $ question <chr> "Op1", "Op1", "Op1", "Op1", "Op1", "Op1", "Op1",
"Op1", "Op1"...
## $ score <dbl> 3, 6, 6, 6, 6, 3, 3, 6, 6, 3, 4, 5, 5, 5, 6, 4, 1,
2, 5, 6, 4...

```

6.5.2 separate()

- `col` is the column you want to separate
- `into` is a vector of new column names
- `sep` is the character(s) that separate your new columns. This defaults to anything that isn't alphanumeric, like `.`, `_`, `-`, `:` and is like the `names_sep` argument in `pivot_longer()`.

Split the `question` column into two columns: `domain` and `qnumber`.

There is no character to split on, here, but you can separate a column after a specific number of characters by setting `sep` to an integer. For example, to split "abcde" after the third character, use `sep = 3`, which results in `c("abc", "de")`. You can also use negative number to split before the *n*th character from the right. For example, to split a column that has words of various lengths and 2-digit suffixes (like "lisa03", "amanda38"), you can use `sep = -2`.

```

personality_sep <- separate(
  data = personality_gathered,
  col = question,          # column to separate
  into = c("domain", "qnumber"), # new column names
  sep = 2                  # where to separate
) %>%
  glimpse()

```

```
## Rows: 615,000
## Columns: 5
## $ user_id <dbl> 0, 1, 2, 5, 8, 108, 233, 298, 426, 436, 685, 807,
871, 881, 94...
## $ date <date> 2006-03-23, 2006-02-08, 2005-10-24, 2005-12-07,
2006-07-27, 2...
## $ domain <chr> "Op", "Op", "Op", "Op", "Op", "Op", "Op", "Op",
"Op", "Op", "0...
## $ qnumber <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1",
"1", "1", "1...
## $ score <dbl> 3, 6, 6, 6, 6, 3, 3, 6, 6, 3, 4, 5, 5, 5, 6, 4, 1,
2, 5, 6, 4,...
```

If you want to separate just at full stops, you need to use `sep = "\\."`, not `sep = "."`. The two slashes **escape** the full stop, making it interpreted as a literal full stop and not the regular expression for any character.

6.5.3 `unite()`

- `col` is your new united column
- `...` refers to the columns you want to unite
- `sep` is the character(s) that will separate your united columns

Put the domain and qnumber columns back together into a new column named `domain_n`. Make it in a format like "Op_Q1".

```
personality_unite <- unite(
  data = personality_sep,
  col = "domain_n", # new column name
  domain, qnumber, # columns to unite
  sep = "_Q"        # separation characters
) %>%
  glimpse()
```

```
## Rows: 615,000
## Columns: 4
## $ user_id <dbl> 0, 1, 2, 5, 8, 108, 233, 298, 426, 436, 685, 807,
871, 881, 9...
## $ date <date> 2006-03-23, 2006-02-08, 2005-10-24, 2005-12-07,
2006-07-27, ...
## $ domain_n <chr> "Op_Q1", "Op_Q1", "Op_Q1", "Op_Q1", "Op_Q1",
"Op_Q1", "Op_Q1"...
## $ score <dbl> 3, 6, 6, 6, 6, 3, 3, 6, 6, 3, 4, 5, 5, 5, 6, 4, 1,
2, 5, 6, 4,...
```

6.5.4 spread()

You can reverse the processes above, as well. For example, you can convert data from long format into wide format.

- `key` is the column that contains your new column headers. It is like `names_from` in `pivot_wider()`, but can only take one value (multiple values need to be merged first using `unite()`).
- `value` is the column that contains the values in the new spread columns. It is like `values_from` in `pivot_wider()`.

```
personality_spread <- spread(
  data = personality_unite,
  key = domain_n, # column that contains new headers
  value = score    # column that contains new values
) %>%
  glimpse()
```

```
## Rows: 15,000
## Columns: 43
## $ user_id <dbl> 0, 1, 2, 5, 8, 108, 233, 298, 426, 436, 685, 807,
## 871, 881, 94...
## $ date    <date> 2006-03-23, 2006-02-08, 2005-10-24, 2005-12-07,
## 2006-07-27, 2...
## $ Ag_Q1   <dbl> 2, 0, 0, 4, 6, 5, 5, 4, 2, 5, 4, 3, 2, 4, 5, 3, 5,
## 5, 5, 4, 4,...
## $ Ag_Q2   <dbl> 1, 6, 6, 0, 5, 4, 5, 3, 4, 3, 5, 1, 5, 4, 2, 6, 5,
## 5, 5, 5, 2,...
## $ Ag_Q3   <dbl> 1, 0, 1, 1, 0, 4, 4, 4, 3, 3, 4, 4, 3, 4, 4, 5, 5,
## 4, 5, 3, 4,...
## $ Ag_Q4   <dbl> 1, 0, 1, 4, 6, 5, 5, 6, 6, 6, 4, 2, 4, 5, 4, 5, 6,
## 4, 5, 6, 5,...
## $ Ag_Q5   <dbl> 3, 6, 5, 0, 2, 5, 6, 2, 2, 3, 4, 1, 3, 5, 2, 6, 5,
## 6, 5, 3, 3,...
## $ Ag_Q6   <dbl> NA, 6, 5, 2, 3, 4, 5, 6, 1, 3, 4, 2, 3, 5, 1, 6, 2,
## 6, 6, 5, 3...
## $ Ag_Q7   <dbl> 3, 0, 1, 1, 1, 3, 3, 5, 0, 3, 2, 1, 2, 3, 5, 6, 4,
## 4, 6, 6, 2,...
## $ Co_Q1   <dbl> 3, 0, 0, 3, 5, 4, 3, 4, 5, 3, 3, 3, 1, 5, 5, 4, 4,
## 5, 6, 4, 2,...
## $ Co_Q10  <dbl> 1, 6, 5, 5, 3, 5, 1, 2, 5, 2, 4, 3, 4, 4, 3, 2, 5,
## 5, 5, 2, 2,...
## $ Co_Q2   <dbl> 3, 0, 0, 3, 4, 3, 3, 4, 5, 3, 5, 3, 3, 4, 5, 1, 5,
## 4, 5, 2, 5,...
## $ Co_Q3   <dbl> 2, 0, 1, 3, 4, 4, 5, 4, 5, 3, 4, 3, 4, 4, 5, 4, 2,
## 4, 5, 2, 2,...
## $ Co_Q4   <dbl> 3, 6, 5, 5, 5, 3, 2, 4, 3, 1, 4, 3, 1, 2, 4, 2, NA,
```

```

5, 6, 1, 1...
## $ Co_Q5 <dbl> 0, 6, 5, 5, 5, 3, 3, 1, 5, 1, 2, 4, 4, 4, 2, 1, 6,
4, 3, 1, 3,...
## $ Co_Q6 <dbl> 6, 0, 1, 4, 6, 5, 6, 5, 4, 3, 5, 5, 4, 6, 6, 1, 3,
4, 5, 4, 6,...
## $ Co_Q7 <dbl> 3, 6, 5, 1, 3, 4, NA, 2, 3, 3, 2, 2, 4, 2, 5, 2, 5,
5, 3, 1, 1...
## $ Co_Q8 <dbl> 3, 0, 1, 1, 3, 4, 3, 0, 1, 3, 2, 2, 1, 2, 4, 3, 2,
4, 5, 2, 6,...
## $ Co_Q9 <dbl> 3, 6, 5, 4, 3, 4, 5, 3, 5, 3, 4, 3, 4, 4, 2, 4, 6,
5, 5, 2, 2,...
## $ Ex_Q1 <dbl> 3, 0, 0, 2, 2, 4, 4, 3, 5, 4, 1, 1, 3, 3, 1, 3, 5,
1, 0, 4, 1,...
## $ Ex_Q2 <dbl> 3, 0, 0, 3, 3, 4, 5, 2, 5, 3, 4, 1, 3, 2, 1, 6, 5,
3, 4, 4, 1,...
## $ Ex_Q3 <dbl> 3, 6, 5, 5, 3, 3, 3, 0, 6, 1, 4, 2, 3, 2, 1, 2, 5,
1, 0, 5, 5,...
## $ Ex_Q4 <dbl> 1, 0, 1, 3, 3, 3, 4, 3, 5, 3, 2, 0, 3, 3, 1, 2, NA,
4, 4, 4, 1...
## $ Ex_Q5 <dbl> 3, 0, 1, 6, 3, 3, 4, 2, 5, 2, 2, 4, 2, 3, 0, 4, 5,
2, 3, 1, 1,...
## $ Ex_Q6 <dbl> 3, 6, 5, 3, 0, 4, 3, 1, 6, 3, 2, 1, 4, 2, 1, 5, 6,
2, 1, 2, 1,...
## $ Ex_Q7 <dbl> 3, 6, 5, 4, 1, 2, 5, 3, 6, 3, 4, 3, 5, 1, 1, 6, 6,
3, 1, 1, 3,...
## $ Ex_Q8 <dbl> 2, 0, 1, 4, 3, 4, 2, 4, 6, 2, 4, 0, 4, 4, 1, 3, 5,
4, 3, 1, 1,...
## $ Ex_Q9 <dbl> 4, 6, 5, 5, 5, 2, 3, 3, 6, 3, 3, 4, 4, 3, 2, 5, 5,
4, 4, 0, 4,...
## $ Ne_Q1 <dbl> 4, 0, 0, 4, 1, 2, 3, 4, 0, 3, 3, 3, 2, 1, 1, 3, 4,
5, 2, 4, 5,...
## $ Ne_Q2 <dbl> 0, 6, 6, 4, 2, 1, 2, 3, 1, 2, 5, 5, 3, 1, 1, 1, 1,
6, 1, 2, 5,...
## $ Ne_Q3 <dbl> 0, 0, 0, 1, 0, 1, 4, 4, 0, 4, 2, 5, 1, 2, 5, 5, 2,
2, 1, 2, 5,...
## $ Ne_Q4 <dbl> 3, 6, 6, 2, 3, 2, 3, 3, 0, 4, 4, 5, 5, 4, 5, 3, 2,
5, 2, 4, 5,...
## $ Ne_Q5 <dbl> 3, 0, 1, 4, 1, 1, 4, 5, 0, 3, 4, 6, 2, 0, 1, 1, 0,
4, 3, 1, 5,...
## $ Ne_Q6 <dbl> 1, 6, 5, 1, 0, 1, 3, 4, 0, 4, 4, 5, 2, 1, 5, 6, 1,
2, 2, 3, 5,...
## $ Ne_Q7 <dbl> NA, 0, 1, 2, 0, 2, 4, 4, 0, 3, 2, 5, 1, 2, 5, 2, 2,
4, 1, 3, 5...
## $ Ne_Q8 <dbl> 2, 0, 1, 1, 1, 1, 5, 4, 0, 4, 4, 5, 1, 2, 5, 2, 1,
5, 1, 2, 5,...
## $ Op_Q1 <dbl> 3, 6, 6, 6, 6, 3, 3, 6, 6, 3, 4, 5, 5, 5, 6, 4, 1,
2, 5, 6, 4,...

```

```
## $ Op_Q2 <dbl> 6, 0, 0, 4, 6, 4, 4, 0, 0, 3, 4, 3, 3, 4, 5, 3, 3,
4, 1, 6, 6,...
## $ Op_Q3 <dbl> 2, 6, 5, 5, 5, 4, 3, 2, 4, 3, 3, 6, 5, 5, 6, 5, 4,
4, 3, 6, 5,...
## $ Op_Q4 <dbl> 3, 0, 1, 6, 6, 3, 3, 0, 6, 3, 4, 5, 4, 5, 6, 6, 2,
2, 4, 5, 5,...
## $ Op_Q5 <dbl> 6, 6, 5, 2, 5, 4, 3, 2, 6, 6, 2, 4, 3, 4, 6, 6, 6,
5, 3, 3, 5,...
## $ Op_Q6 <dbl> 0, 6, 5, 1, 6, 4, 6, 0, 0, 3, 5, 3, 5, 5, 5, 2, 5,
1, 1, 6, 2,...
## $ Op_Q7 <dbl> 0, 6, 5, 5, 5, 4, 6, 2, 1, 3, 2, 4, 5, 5, 6, 3, 6,
5, 2, 6, 5,...
```

6.6 Pipes

Pipes are a way to order your code in a more readable format.

Let's say you have a small data table with 10 participant IDs, two columns with variable type A, and 2 columns with variable type B. You want to calculate the mean of the A variables and the mean of the B variables and return a table with 10 rows (1 for each participant) and 3 columns (`id` , `A_mean` and `B_mean`).



One way you could do this is by creating a new object at every step and using that object in the next step. This is pretty clear, but you've created 6 unnecessary data objects in your environment. This can get confusing in very long scripts.

```

# make a data table with 10 subjects
data_original <- tibble(
  id = 1:10,
  A1 = rnorm(10, 0),
  A2 = rnorm(10, 1),
  B1 = rnorm(10, 2),
  B2 = rnorm(10, 3)
)

# gather columns A1 to B2 into "variable" and "value" columns
data_gathered <- gather(data_original, variable, value, A1:B2)

# separate the variable column at the _ into "var" and "var_n" columns
data_separated <- separate(data_gathered, variable, c("var", "var_n"),
  sep = 1)

# group the data by id and var
data_grouped <- group_by(data_separated, id, var)

# calculate the mean value for each id/var
data_summarised <- summarise(data_grouped, mean = mean(value), .groups
  = "drop")

# spread the mean column into A and B columns
data_spread <- spread(data_summarised, var, mean)

# rename A and B to A_mean and B_mean
data <- rename(data_spread, A_mean = A, B_mean = B)

data

```

id	A_mean	B_mean
1	-0.5938256	1.0243046
2	0.7440623	2.7172046
3	0.9309275	3.9262358
4	0.7197686	1.9662632
5	-0.0280832	1.9473456
6	-0.0982555	3.2073687
7	0.1256922	0.9256321
8	1.4526447	2.3778116
9	0.2976443	1.6617481

10

0.5589199

2.1034679

You *can* name each object `data` and keep replacing the old data object with the new one at each step. This will keep your environment clean, but I don't recommend it because it makes it too easy to accidentally run your code out of order when you are running line-by-line for development or debugging.

One way to avoid extra objects is to nest your functions, literally replacing each data object with the code that generated it in the previous step. This can be fine for very short chains.

```
mean_petal_width <- round(mean(iris$Petal.Width), 2)
```

But it gets extremely confusing for long chains:

```
# do not ever do this!!
data <- rename(
  spread(
    summarise(
      group_by(
        separate(
          gather(
            tibble(
              id = 1:10,
              A1 = rnorm(10, 0),
              A2 = rnorm(10, 1),
              B1 = rnorm(10, 2),
              B2 = rnorm(10, 3)),
            variable, value, A1:B2),
            variable, c("var", "var_n"), sep = 1),
            id, var),
          mean = mean(value), .groups = "drop"),
        var, mean),
    A_mean = A, B_mean = B)
```

The pipe lets you "pipe" the result of each function into the next function, allowing you to put your code in a logical order without creating too many extra objects.

```
# calculate mean of A and B variables for each participant
data <- tibble(
  id = 1:10,
  A1 = rnorm(10, 0),
  A2 = rnorm(10, 1),
  B1 = rnorm(10, 2),
  B2 = rnorm(10, 3)
) %>%
gather(variable, value, A1:B2) %>%
separate(variable, c("var", "var_n"), sep=1) %>%
group_by(id, var) %>%
summarise(mean = mean(value), .groups = "drop") %>%
spread(var, mean) %>%
rename(A_mean = A, B_mean = B)
```

You can read this code from top to bottom as follows:

1. Make a tibble called `data` with
 - id of 1 to 10,
 - A1 of 10 random numbers from a normal distribution,
 - A2 of 10 random numbers from a normal distribution,
 - B1 of 10 random numbers from a normal distribution,
 - B2 of 10 random numbers from a normal distribution; and then
2. Gather to create `variable` and `value` column from columns `A_1` to `B_2`; and then
3. Separate the column `variable` into 2 new columns called `var` and `var_n`, separate at character 1; and then
4. Group by columns `id` and `var`; and then
5. Summarise and new column called `mean` as the mean of the `value` column for each group and drop the grouping; and then
6. Spread to make new columns with the key names in `var` and values in `mean`; and then
7. Rename to make columns called `A_mean` (old `A`) and `B_mean` (old `B`)

You can make intermediate objects whenever you need to break up your code because it's getting too complicated or you need to debug something.

You can debug a pipe by highlighting from the beginning to just before the pipe you want to stop at. Try this by highlighting from `data <-` to the end of the `separate` function and typing `cmd-return`. What does `data` look like now?

Chain all the steps above using pipes.

```
personality_reshaped <- personality %>%
  gather("question", "score", 0p1:Ex9) %>%
  separate(question, c("domain", "qnumber"), sep = 2) %>%
  unite("domain_n", domain, qnumber, sep = "_Q") %>%
  spread(domain_n, score)
```

6.7 More Complex Example

6.7.1 Load Data

Get data on infant and maternal mortality rates from the reprores package. If you don't have the package, you can download them here:

- [infant mortality](#)
- [maternal mortality](#)

```
data("infmort")
head(infmort)
```

Infant mortality rate (probability of dying between birth and			
Country	Year	age 1 per 1000 live births)	
Afghanistan	2015	66.3 [52.7-83.9]	
Afghanistan	2014	68.1 [55.7-83.6]	
Afghanistan	2013	69.9 [58.7-83.5]	
Afghanistan	2012	71.7 [61.6-83.7]	
Afghanistan	2011	73.4 [64.4-84.2]	
Afghanistan	2010	75.1 [66.9-85.1]	

```
data("matmort")
head(matmort)
```

Country	1990	2000	2015
Afghanistan	1 340 [878 - 1 950]	1 100 [745 - 1 570]	396 [253 - 620]
Albania	71 [58 - 88]	43 [33 - 56]	29 [16 - 46]
Algeria	216 [141 - 327]	170 [118 - 241]	140 [82 - 244]
Angola	1 160 [627 - 2 020]	924 [472 - 1 730]	477 [221 - 988]

Argentina	72 [64 - 80]	60 [54 - 65]	52 [44 - 63]
Armenia	58 [51 - 65]	40 [35 - 46]	25 [21 - 31]

6.7.2 Wide to Long

`matmort` is in wide format, with a separate column for each year. Change it to long format, with a row for each Country/Year observation.

This example is complicated because the column names to gather *are* numbers. If the column names are non-standard (e.g., have spaces, start with numbers, or have special characters), you can enclose them in backticks (```) like the example below.

```
matmort_long <- matmort %>%
  pivot_longer(cols = `1990`:`2015`,
               names_to = "Year",
               values_to = "stats") %>%
  glimpse()
```

```
## Rows: 543
## Columns: 3
## $ Country <chr> "Afghanistan", "Afghanistan", "Afghanistan",
"Albania", "Alban...
## $ Year <chr> "1990", "2000", "2015", "1990", "2000", "2015",
"1990", "2000"...
## $ stats <chr> "1 340 [ 878 - 1 950]", "1 100 [ 745 - 1 570]", "396
[ 253 - ...
```

You can put `matmort` at the first argument to `pivot_longer()`; you don't have to pipe it in. But when I'm working on data processing I often find myself needing to insert or rearrange steps and I constantly introduce errors by forgetting to take the first argument out of a pipe chain, so now I start with the original data table and pipe from there.

Alternatively, you can use the `gather()` function.

```
matmort_long <- matmort %>%
  gather("Year", "stats", `1990`:`2015`) %>%
  glimpse()
```

```
## Rows: 543
## Columns: 3
## $ Country <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina", "A...
## $ Year <chr> "1990", "1990", "1990", "1990", "1990", "1990",
"1990", "1990"...
## $ stats <chr> "1 340 [ 878 - 1 950]", "71 [ 58 - 88]", "216 [ 141
- 327]",...
```

6.7.3 One Piece of Data per Column

The data in the `stats` column is in an unusual format with some sort of confidence interval in brackets and lots of extra spaces. We don't need any of the spaces, so first we'll remove them with `mutate()`, which we'll learn more about in the next lesson.

The `separate` function will separate your data on anything that is not a number or letter, so try it first without specifying the `sep` argument. The `into` argument is a list of the new column names.

```
matmort_split <- matmort_long %>%
  mutate(stats = gsub(" ", "", stats)) %>%
  separate(stats, c("rate", "ci_low", "ci_hi")) %>%
  glimpse()
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 543 rows
[1, 2, 3, 4,
## 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

```
## Rows: 543
## Columns: 5
## $ Country <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina", "A...
## $ Year <chr> "1990", "1990", "1990", "1990", "1990", "1990",
"1990", "1990"...
## $ rate <chr> "1340", "71", "216", "1160", "72", "58", "8", "8",
"64", "46",...
## $ ci_low <chr> "878", "58", "141", "627", "64", "51", "7", "7",
"56", "34", "...
## $ ci_hi <chr> "1950", "88", "327", "2020", "80", "65", "9", "10",
"74", "61"...
```

The `gsub(pattern, replacement, x)` function is a flexible way to do search and replace. The example above replaces all occurrences of the pattern " " (a space), with the replacement "" (nothing), in the string `x`

(the `stats` column). Use `sub()` instead if you only want to replace the first occurrence of a pattern. We only used a simple pattern here, but you can use more complicated [regex](#) patterns to replace, for example, all even numbers (e.g., `gsub("[:02468:]", "*", "id = 123456")`) or all occurrences of the word colour in US or UK spelling (e.g., `gsub("colo(u)?r", "***", "replace color, colour, or colours, but not collors")`).

6.7.3.1 Handle spare columns with `extra`

The previous example should have given you an error warning about "Additional pieces discarded in 543 rows". This is because `separate` splits the column at the brackets and dashes, so the text `100[90-110]` would split into four values `c("100", "90", "110", "")`, but we only specified 3 new columns. The fourth value is always empty (just the part after the last bracket), so we are happy to drop it, but `separate` generates a warning so you don't do that accidentally. You can turn off the warning by adding the `extra` argument and setting it to "drop". Look at the help for `tidyr::separate` to see what the other options do.

```
matmort_split <- matmort_long %>%
  mutate(stats = gsub(" ", "", stats)) %>%
  separate(stats, c("rate", "ci_low", "ci_hi"), extra = "drop") %>%
  glimpse()
```

```
## Rows: 543
## Columns: 5
## $ Country <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina", "A...
## $ Year <chr> "1990", "1990", "1990", "1990", "1990", "1990",
"1990", "1990"...
## $ rate <chr> "1340", "71", "216", "1160", "72", "58", "8", "8",
"64", "46",...
## $ ci_low <chr> "878", "58", "141", "627", "64", "51", "7", "7",
"56", "34", "...
## $ ci_hi <chr> "1950", "88", "327", "2020", "80", "65", "9", "10",
"74", "61"...
```

6.7.3.2 Set delimiters with `sep`

Now do the same with `infmtort`. It's already in long format, so you don't need to use `gather`, but the third column has a ridiculously long name, so we can just refer to it by its column number (3).

```
infmort_split <- infmort %>%
  separate(3, c("rate", "ci_low", "ci_hi"), extra = "drop") %>%
  glimpse()
```

```
## Rows: 5,044
## Columns: 5
## $ Country <chr> "Afghanistan", "Afghanistan", "Afghanistan",
"Afghanistan", "A...
## $ Year <dbl> 2015, 2014, 2013, 2012, 2011, 2010, 2009, 2008,
2007, 2006, 20...
## $ rate <chr> "66", "68", "69", "71", "73", "75", "76", "78",
"80", "82", "8...
## $ ci_low <chr> "3", "1", "9", "7", "4", "1", "8", "6", "4", "3",
"4", "7", "0...
## $ ci_hi <chr> "52", "55", "58", "61", "64", "66", "69", "71",
"73", "75", "7..."
```

Wait, that didn't work at all! It split the column on spaces, brackets, *and* full stops. We just want to split on the spaces, brackets and dashes. So we need to manually set `sep` to what the delimiters are. Also, once there are more than a few arguments specified for a function, it's easier to read them if you put one argument on each line.

You can use [regular expressions](#) to separate complex columns. Here, we want to separate on dashes and brackets. You can separate on a list of delimiters by putting them in parentheses, separated by "|". It's a little more complicated because brackets have a special meaning in regex, so you need to "escape" the left one with two backslashes "\\".

```
infmort_split <- infmort %>%
  separate(
    col = 3,
    into = c("rate", "ci_low", "ci_hi"),
    extra = "drop",
    sep = "(\\[|-|])"
  ) %>%
  glimpse()
```

```
## Rows: 5,044
## Columns: 5
## $ Country <chr> "Afghanistan", "Afghanistan", "Afghanistan",
"Afghanistan", "A...
## $ Year <dbl> 2015, 2014, 2013, 2012, 2011, 2010, 2009, 2008,
2007, 2006, 20...
## $ rate <chr> "66.3 ", "68.1 ", "69.9 ", "71.7 ", "73.4 ", "75.1
", "76.8 ",...
## $ ci_low <chr> "52.7", "55.7", "58.7", "61.6", "64.4", "66.9",
"69.0", "71.2"...
## $ ci_hi <chr> "83.9", "83.6", "83.5", "83.7", "84.2", "85.1",
"86.1", "87.3"...
```

6.7.3.3 Fix data types with `convert`

That's better. Notice the next to `Year`, `rate`, `ci_low` and `ci_hi`. That means these columns hold characters (like words), not numbers or integers. This can cause problems when you try to do things like average the numbers (you can't average words), so we can fix it by adding the argument `convert` and setting it to `TRUE`.

```
infmtort_split <- infmort %>%
  separate(col = 3,
    into = c("rate", "ci_low", "ci_hi"),
    extra = "drop",
    sep = "\\[|-|]",
    convert = TRUE) %>%
  glimpse()
```

```
## Rows: 5,044
## Columns: 5
## $ Country <chr> "Afghanistan", "Afghanistan", "Afghanistan",
"Afghanistan", "A...
## $ Year <dbl> 2015, 2014, 2013, 2012, 2011, 2010, 2009, 2008,
2007, 2006, 20...
## $ rate <dbl> 66.3, 68.1, 69.9, 71.7, 73.4, 75.1, 76.8, 78.6,
80.4, 82.3, 84...
## $ ci_low <dbl> 52.7, 55.7, 58.7, 61.6, 64.4, 66.9, 69.0, 71.2,
73.4, 75.5, 77...
## $ ci_hi <dbl> 83.9, 83.6, 83.5, 83.7, 84.2, 85.1, 86.1, 87.3,
88.9, 90.7, 92...
```

Do the same for `matmort`.

Table of contents

[Overview](#)

[1 Getting Started](#)

[2 Reproducible Workflows](#)

[3 Data Visualisation](#)

[4 Working with Data](#)

[5 Data Relations](#)

[6 Tidy Data](#)

[7 Data Wrangling](#)

[8 Introduction to GLM](#)

[9 Iteration & Functions](#)

[10 Probability & Simulation](#)

[Acknowledgements](#)

Appendices

[A Installing and Updating](#)

[B Conventions](#)[C Symbols](#)[D Datasets](#)[E Styling Plots](#)[F Exercises](#)[G References](#)[View book source](#) 

```
matmort_split <- matmort_long %>%
  mutate(stats = gsub(" ", "", stats)) %>%
  separate(col = stats,
           into = c("rate", "ci_low", "ci_hi"),
           extra = "drop",
           convert = TRUE) %>%
  glimpse()
```

```
## Rows: 543
## Columns: 5
## $ Country <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina", "A...
## $ Year <chr> "1990", "1990", "1990", "1990", "1990", "1990",
"1990", "1990"...
## $ rate <int> 1340, 71, 216, 1160, 72, 58, 8, 8, 64, 46, 26, 569,
58, 33, 9,...
## $ ci_low <int> 878, 58, 141, 627, 64, 51, 7, 7, 56, 34, 20, 446,
47, 28, 7, 4...
## $ ci_hi <int> 1950, 88, 327, 2020, 80, 65, 9, 10, 74, 61, 33, 715,
72, 38, 1...
```

6.7.4 All in one step

We can chain all the steps for `matmort` above together, since we don't need those intermediate data tables.

```
matmort2 <- matmort %>%
  gather("Year", "stats", `1990`:`2015`) %>%
  mutate(stats = gsub(" ", "", stats)) %>%
  separate(
    col = stats,
    into = c("rate", "ci_low", "ci_hi"),
    extra = "drop",
    convert = TRUE
  ) %>%
  glimpse()
```

```
## Rows: 543
## Columns: 5
## $ Country <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina", "A...
## $ Year <chr> "1990", "1990", "1990", "1990", "1990", "1990",
"1990", "1990"...
## $ rate <int> 1340, 71, 216, 1160, 72, 58, 8, 8, 64, 46, 26, 569,
58, 33, 9,...
## $ ci_low <int> 878, 58, 141, 627, 64, 51, 7, 7, 56, 34, 20, 446,
47, 28, 7, 4...
## $ ci_hi <int> 1950, 88, 327, 2020, 80, 65, 9, 10, 74, 61, 33, 715,
72, 38, 1...
```

6.7.5 Columns by Year

Spread out the maternal mortality rate by year.

```
matmort_wide <- matmort2 %>%
  spread(key = Year, value = rate) %>%
  print()
```

```
## # A tibble: 542 × 6
##   Country      ci_low ci_hi `1990` `2000` `2015`
##   <chr>      <int> <int> <int> <int> <int>
## 1 Afghanistan    253   620    NA     NA   396
## 2 Afghanistan    745  1570    NA  1100    NA
## 3 Afghanistan    878  1950   1340    NA    NA
## 4 Albania         16    46    NA     NA   29
## 5 Albania         33    56    NA    43    NA
## 6 Albania         58    88    71    NA    NA
## 7 Algeria         82   244    NA    NA   140
## 8 Algeria        118   241    NA   170    NA
## 9 Algeria        141   327   216    NA    NA
## 10 Angola         221   988    NA    NA   477
## # ... with 532 more rows
```

Nope, that didn't work at all, but it's a really common mistake when spreading data. This is because `spread` matches on all the remaining columns, so Afghanistan with `ci_low` of 253 is treated as a different observation than Afghanistan with `ci_low` of 745.

This is where `pivot_wider()` can be very useful. You can set `values_from` to multiple column names and their names will be added to the `names_from` values.


```
matmortWide <- matmort2 %>%
  pivot_wider(
    names_from = Year,
    values_from = c(rate, ci_low, ci_hi)
  )

glimpse(matmortWide)
```

```
## Rows: 181
## Columns: 10
## $ Country      <chr> "Afghanistan", "Albania", "Algeria", "Angola",
"Argentina"...
## $ rate_1990    <int> 1340, 71, 216, 1160, 72, 58, 8, 8, 64, 46, 26,
569, 58, 33...
## $ rate_2000    <int> 1100, 43, 170, 924, 60, 40, 9, 5, 48, 61, 21,
399, 48, 26,...
## $ rate_2015    <int> 396, 29, 140, 477, 52, 25, 6, 4, 25, 80, 15,
176, 27, 4, 7...
## $ ci_low_1990  <int> 878, 58, 141, 627, 64, 51, 7, 7, 56, 34, 20,
446, 47, 28, ...
## $ ci_low_2000  <int> 745, 33, 118, 472, 54, 35, 8, 4, 42, 50, 18,
322, 38, 22, ...
## $ ci_low_2015  <int> 253, 16, 82, 221, 44, 21, 5, 3, 17, 53, 12, 125,
19, 3, 5,...
## $ ci_hi_1990   <int> 1950, 88, 327, 2020, 80, 65, 9, 10, 74, 61, 33,
715, 72, 3...
## $ ci_hi_2000   <int> 1570, 56, 241, 1730, 65, 46, 10, 6, 55, 74, 26,
496, 58, 3...
## $ ci_hi_2015   <int> 620, 46, 244, 988, 63, 31, 7, 5, 35, 124, 19,
280, 37, 6, ...
```

6.7.6 Experimentum Data

Students in the Institute of Neuroscience and Psychology at the University of Glasgow can use the online experiment builder platform, [Experimentum](#). The platform is also [open source on github](#) for anyone who can install it on a web server. It allows you to group questionnaires and experiments into **projects** with randomisation and counterbalancing. Data for questionnaires and experiments are downloadable in long format, but researchers often need to put them in wide format for analysis.

Look at the help menu for built-in dataset `experimentum_questions` to learn what each column is. Subjects are asked questions about dogs to test the different questionnaire response types.

- current: Do you own a dog? (yes/no)

- past: Have you ever owned a dog? (yes/no)
- name: What is the best name for a dog? (free short text)
- good: How good are dogs? (1=pretty good:7=very good)
- country: What country do borzois come from?
- good_borzois: How good are borzois? (0=pretty good:100=very good)
- text: Write some text about dogs. (free long text)
- time: What time is it? (time)

To get the dataset into wide format, where each question is in a separate column, use the following code:

```
q <- experimentum_quests %>%  
  pivot_wider(id_cols = session_id:user_age,  
              names_from = q_name,  
              values_from = dv) %>%  
  type.convert(as.is = TRUE) %>%  
  print()
```

```
## # A tibble: 24 × 15
##   session_id project_id quest_id user_id user_sex user_status
##   user_age current
##           <int>      <int>      <int>      <int> <chr>      <chr>
<dbl>   <int>
## 1      34034          1          1    31105 female    guest
28.2      1
## 2      34104          1          1    31164 male      registered
19.4      1
## 3      34326          1          1    31392 female    guest
17         0
## 4      34343          1          1    31397 male      guest
22         1
## 5      34765          1          1    31770 female    guest
44         1
## 6      34796          1          1    31796 female    guest
35.9      0
## 7      34806          1          1    31798 female    guest
35         0
## 8      34822          1          1    31802 female    guest
58         1
## 9      34864          1          1    31820 male      guest
20         0
## 10     35014          1          1    31921 female    student
39.2      1
## # ... with 14 more rows, and 7 more variables: past <int>, name <chr>,
## #   good <int>, country <chr>, text <chr>, good_borzoi <int>, time
## #   <chr>
```

The responses in the `dv` column have multiple types (e.g., [integer](#), [double](#), and [character](#)), but they are all represented as character strings when they're in the same column. After you spread the data to wide format, each column should be given the ocrrect data type. The function `type.convert()` makes a best guess at what type each new column should be and converts it. The argument `as.is = TRUE` converts columns where none of the numbers have decimal places to integers.

6.8 Glossary

term	definition
character	A data type representing strings of text.

<u>double</u>	A data type representing a real decimal number
<u>integer</u>	A data type representing whole numbers.
<u>long</u>	Data where each observation is on a separate row
<u>observation</u>	All of the data about a single trial or question.
<u>value</u>	A single number or piece of data.
<u>variable</u>	A word that identifies and stores the value of some data for later use.
<u>wide</u>	Data where all of the observations about one subject are in the same row

6.9 Further Resources

- [Tidy Data](#)
- [Chapter 12: Tidy Data](#) in *R for Data Science*
- [Chapter 18: Pipes](#) in *R for Data Science*
- [Data wrangling cheat sheet](#)

[« 5 Data Relations](#)
[7 Data Wrangling »](#)

On this page

[6 Tidy Data](#)
[6.1 Learning Objectives](#)
[Basic](#)
[Intermediate](#)
[6.2 Setup](#)
[6.3 Tidy Data](#)
[6.3.1 Three Rules](#)
[6.3.2 Wide versus long](#)
[6.4 Pivot Functions](#)
[6.4.1 Load Data](#)
[6.4.2](#)
[pivot_longer\(\)](#)
[6.4.3](#)

[pivot_wider\(\)](#)

[6.5 Tidy Verbs](#)

[6.5.1 gather\(\)](#)

[6.5.2 separate\(\)](#)

[6.5.3 unite\(\)](#)

[6.5.4 spread\(\)](#)

[6.6 Pipes](#)

[6.7 More Complex Example](#)

[6.7.1 Load Data](#)

"Data Skills for Reproducible Research:

DOI 10.5281/zenodo.6527194" was written by . It was last built on 2022-05-07.

#PSYTEACHR REPRODUCIBLE RESEARCH

